Reinforcement learning
Episode 2

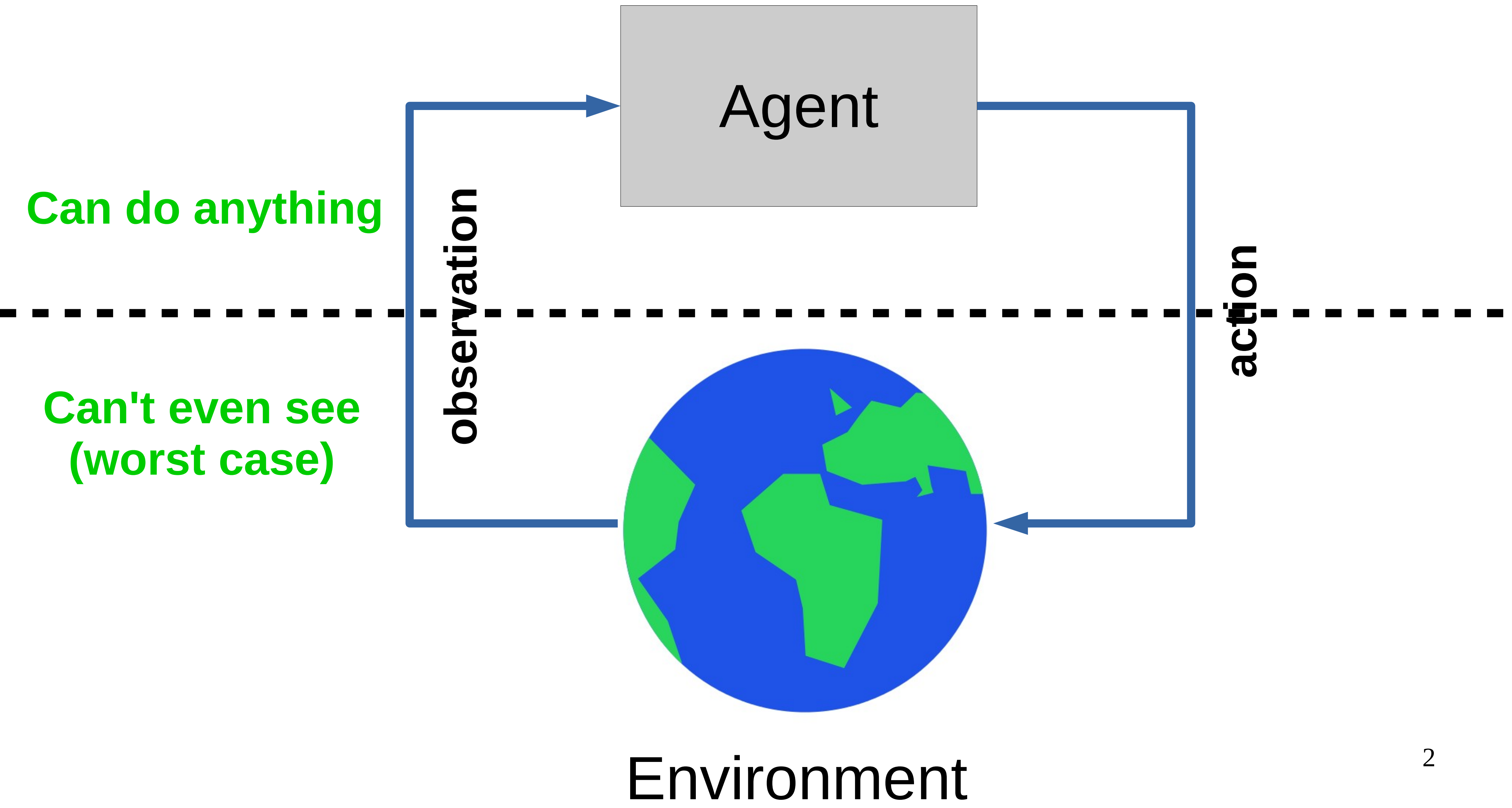
Temporal Difference

Yandex
Data Factory

LAMBDA

British Hedgehog
Preservation Society

Recap: reinforcement learning



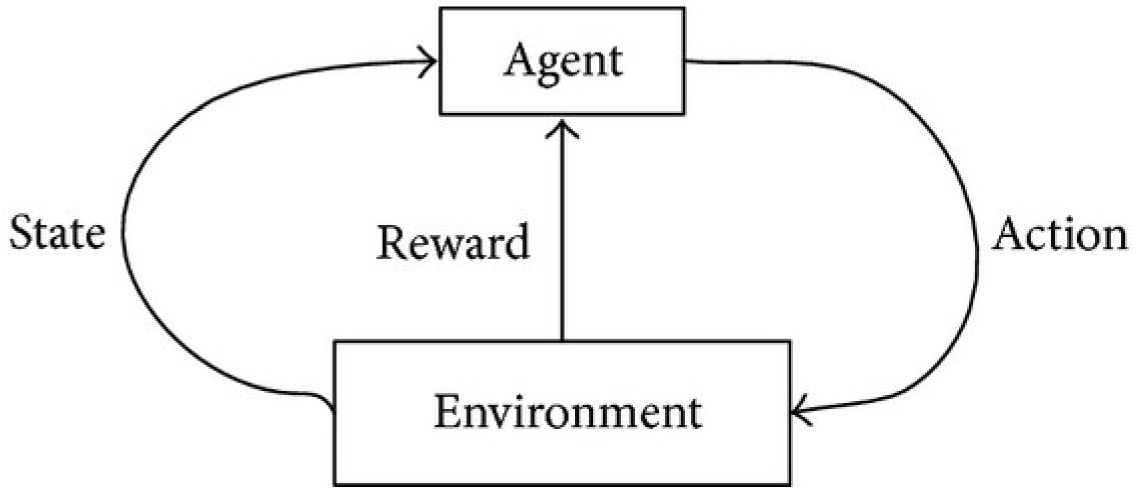
Monte-carlo methods

- R(z) – evaluated at the very end
- Metaheuristics (genetic algorithm, etc.)
- Stochastic optimization (crossentropy method)

Monte-carlo: drawbacks

- Both need a full session to start learning
- Requires a lot of interaction
  - A lot of crashed robots / simulations

MDP formalism: reward on each tick



Classic MDP(Markov Decision Process)
Agent interacts with environment

- Environment states: $s \in S$
- Agent actions: $a \in A$
- State transition: $P(s_{t+1} | s_t, a_t)$
- Reward: $r_t = r(s_t, a_t)$

Discounted reward MDP

Objective:
Total action value

$$R_t = r_t + \gamma \cdot r_{t+1} + \gamma^2 \cdot r_{t+2} + \ldots + \gamma^n \cdot r_{t+n}$$

$$R_t = \sum_i \gamma^i \cdot r_{t+i} \qquad \gamma \in (0,1)\, const$$

γ ~ patience
Cake tomorrow is γ as good as now

Reinforcement learning:
- Find policy that maximizes the expected reward

$$\pi = P(a|s) : E[R] \rightarrow max$$

Discounted reward MDP

Objective:

Total action value

$$R_t = r_t + \gamma \cdot r_{t+1} + \gamma^2 \cdot r_{t+2} + ... + \gamma^n \cdot r_{t+n}$$

$$R_t = \sum_i \gamma^i \cdot r_{t+i} \qquad \gamma \in (0,1)\, const$$

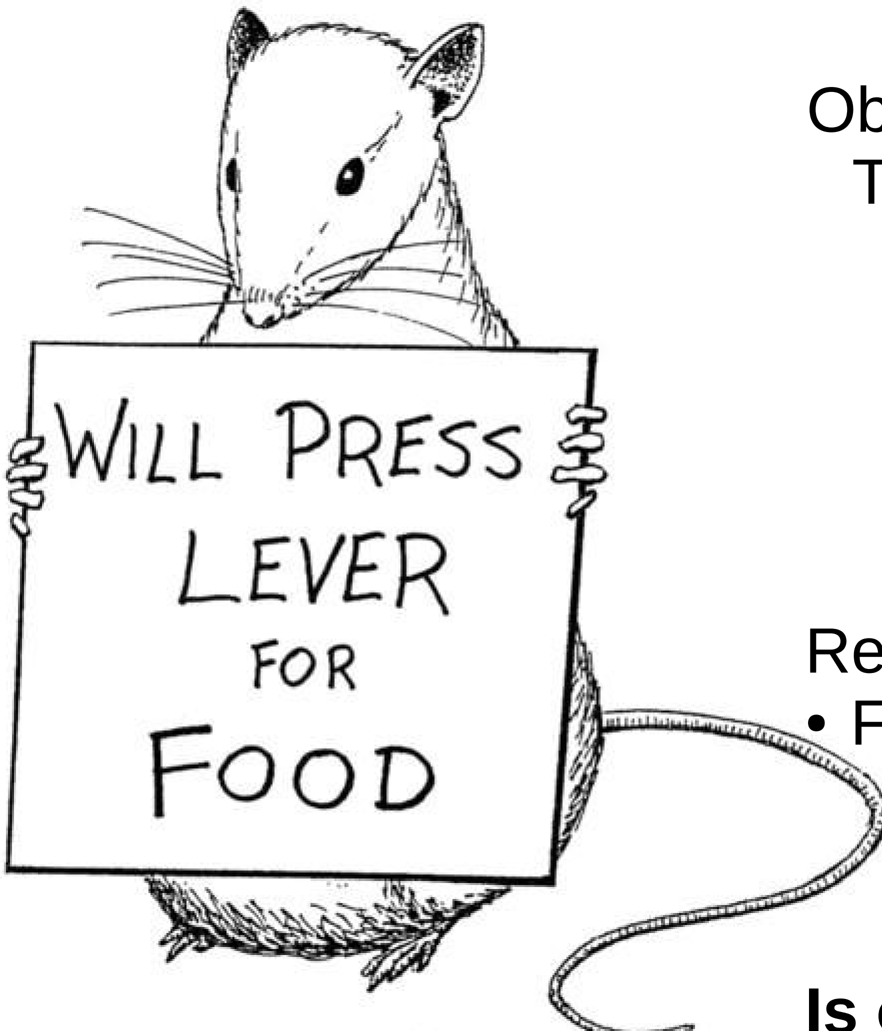
Trivia: which γ corresponds to “only current reward matters”?

Reinforcement learning:

- Find policy that maximizes the expected reward

$$\pi = P(a|s) : E[R] \rightarrow max$$

Discounted reward MDP



Objective:
Total reward

$$R_t = r_t + \gamma \cdot r_{t+1} + \gamma^2 \cdot r_{t+2} + \ldots + \gamma^n \cdot r_{t+n}$$

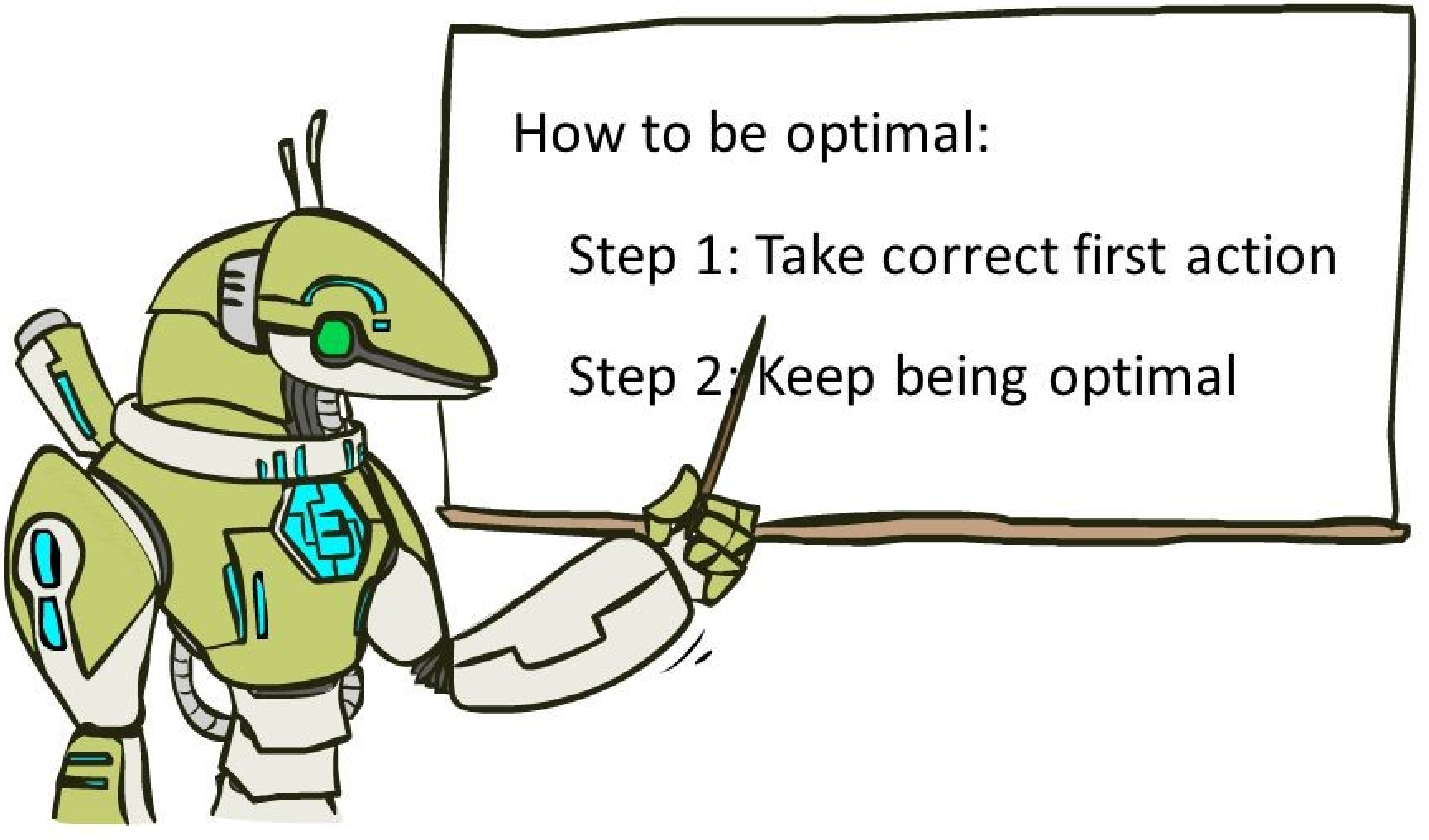
$$R_t = \sum_i \gamma^i \cdot r_{t+i} \qquad \gamma \in (0,1)\, const$$

Reinforcement learning:
- Find policy that maximizes the expected reward

$$\pi = P(a|s) : E[R] \rightarrow max$$

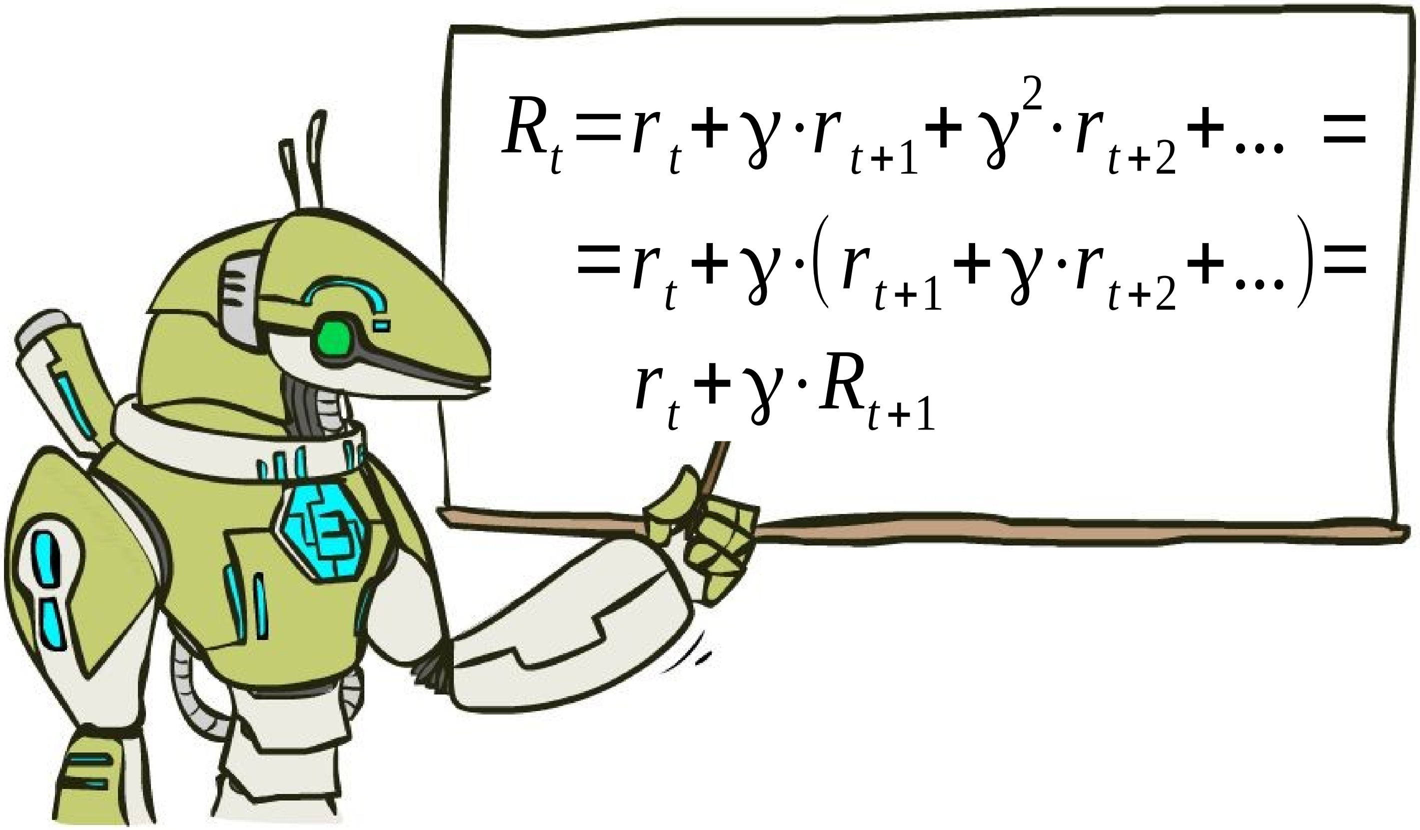
Is optimal policy same as it would be in monte-carlo (if we add-up all r_t)?

Optimal policy



Recurrent optimal strategy definition

Optimal policy



$$R_t = r_t + \gamma \cdot r_{t+1} + \gamma^2 \cdot r_{t+2} + \ldots =$$

$$= r_t + \gamma \cdot \left(r_{t+1} + \gamma \cdot r_{t+2} + \ldots \right) =$$

$$r_t + \gamma \cdot R_{t+1}$$

We rewrite R with sheer power of math!

Value iteration (Temporal Difference)

Idea:
- For each state, obtain V(state)

$$V(s) = max_a [\underbrace{r(s,a)}_{\text{s에서 a를 할 때 reward}} + \gamma \cdot \underbrace{V(s'(s,a))}_{\text{s,a를 통해 나온 새로운 state s'의 Value function}}]$$

Definition V(s) – expected total reward R that can be obtained starting from state s under optimal policy

V(s) : state s부터 시작해서 최적의 policy를 따라 전체 보상 R의 기대값

Value iteration (Temporal Difference)

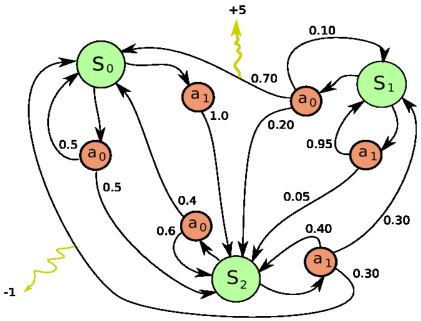
Idea:
- For each state, obtain V(state)

$$V(s) = max_a \left[r(s,a) + \gamma \cdot E_{s' \sim P(s'|s,a)} V(s') \right]$$

근데 사실 state transition
은 확률임..

Stochastic
action outcome

Trivia: if we know the exact V(s) for all states,
how do we determine the best actions?



Value iteration (TD)

Idea:
- Iterative updates

$$\forall s, V_0(s) := 0$$

$$V_{i+1}(s) := max_a [r(s,a) + \gamma \cdot E_{s' \sim P(s'|s,a)} V_i(s')]$$

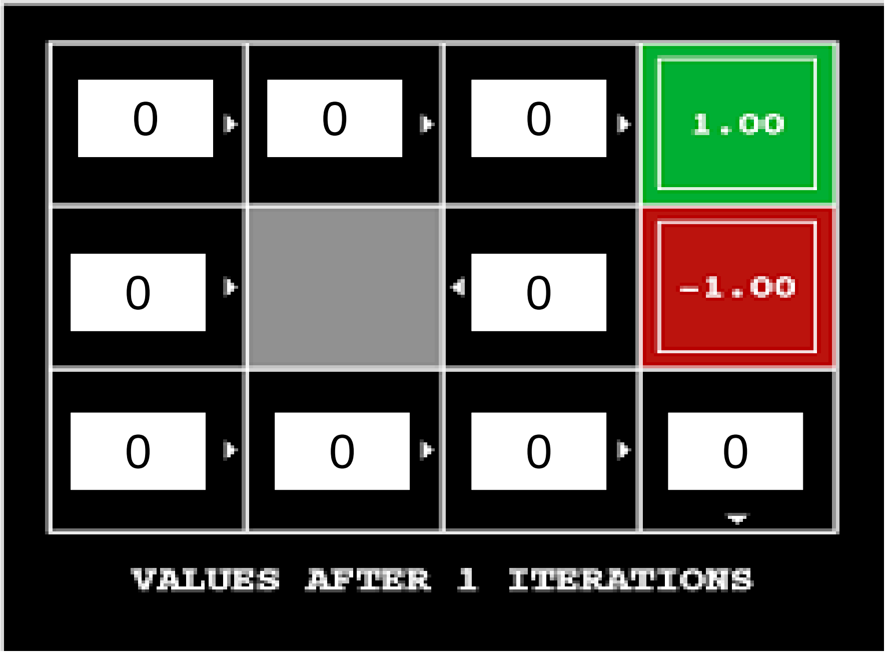
Value iteration (TD)

Idea:

- Iterative updates

$$\forall s, V_0(s) := 0$$

$$V_{i+1}(s) := max_a [r(s,a) + \gamma \cdot E_{s' \sim P(s'|s,a)} V_i(s')]$$



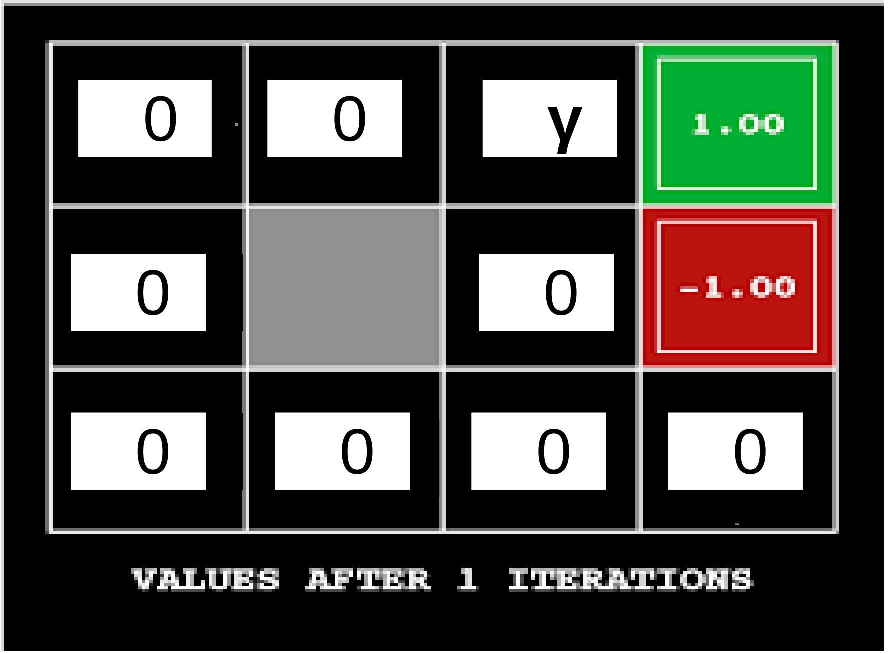
Value iteration (TD)

Idea:

- Iterative updates

$$\forall s, V_0(s) := 0$$

$$V_{i+1}(s) := \max_a [r(s,a) + \gamma \cdot E_{s' \sim P(s'|s,a)} V_i(s')]$$



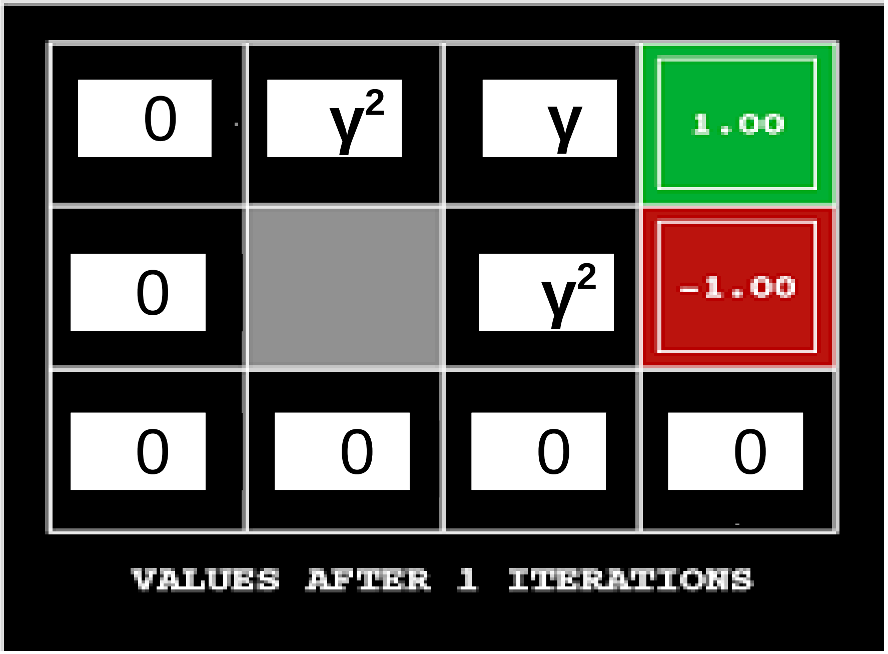
Value iteration (TD)

Idea:

- Iterative updates

$$\forall s, V_0(s) := 0$$

$$V_{i+1}(s) := max_a [r(s,a) + \gamma \cdot E_{s' \sim P(s'|s,a)} V_i(s')]$$



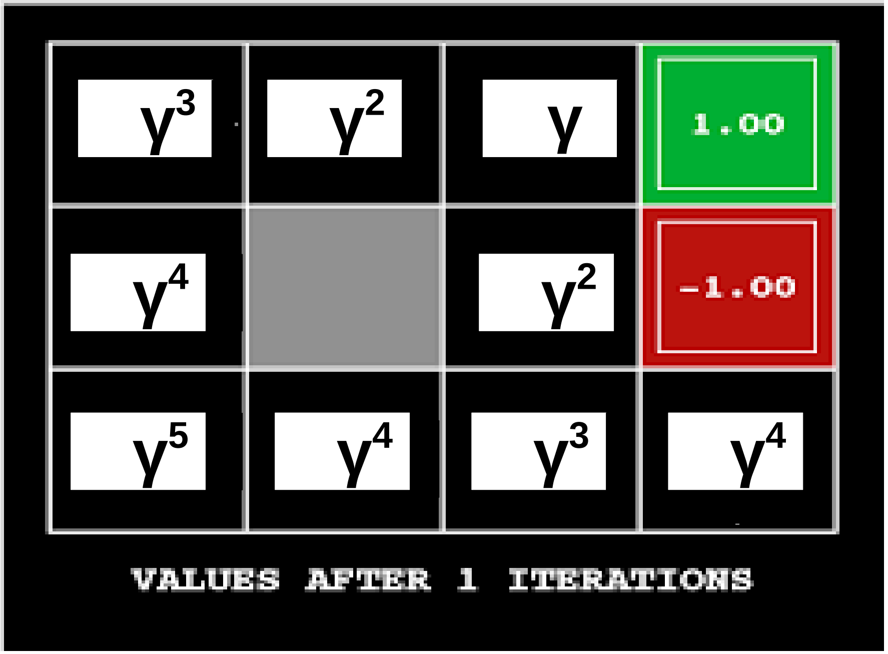
Value iteration (TD)

Idea:

- Iterative updates

$$\forall s, V_0(s) := 0$$

$$V_{i+1}(s) := max_a [r(s,a) + \gamma \cdot E_{s' \sim P(s'|s,a)} V_i(s')]$$



Value iteration (TD)

Idea:
- Iterative updates

$$\forall s, V_0(s) := 0$$

Value를 모두 구하면 인접한 state중 가장
Value가 큰 쪽으로 Policy를 정하면 풀린다.

$$V_{i+1}(s) := max_a \left[r(s,a) + \gamma \cdot E_{s' \sim P(s'|s,a)} V_i(s') \right]$$

* P(s'ls,a)가 달라서 Value가 다름

Voila! We've solved the reinforcement learning!

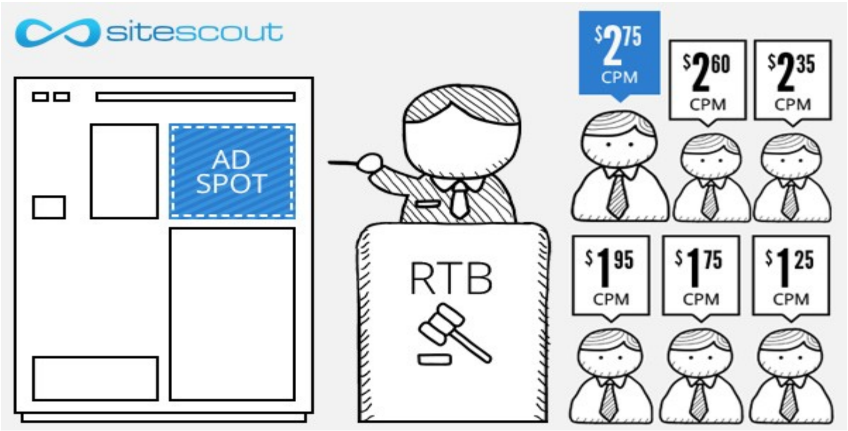
Voila! We've solved the reinforcement learning!

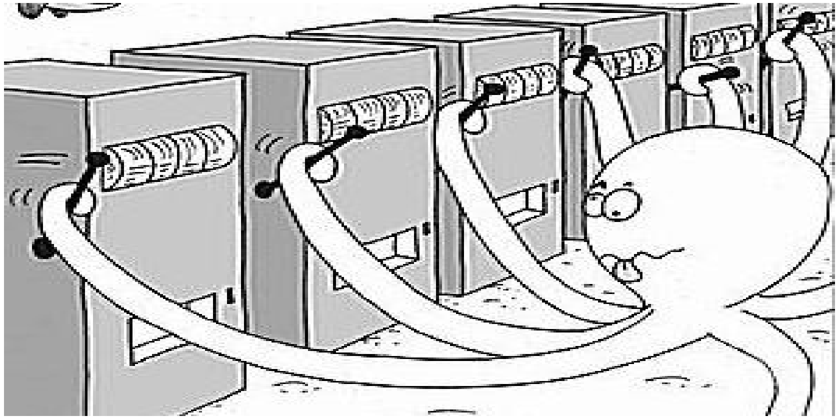
Or have we?

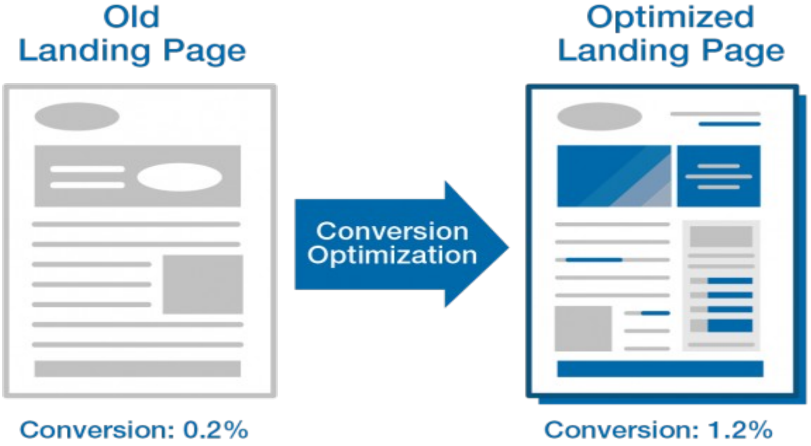
What happens if we apply it to real world problems?

Reality check: web

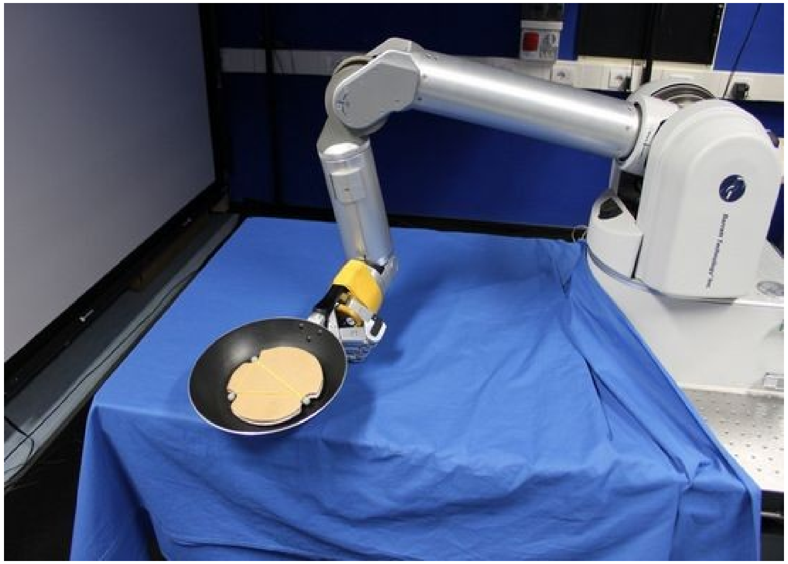
- **Cases:**
  - Pick ads to maximize profit
  - Design landing page to maximize user retention
  - Recommend items to users
- **Common traits:**
  - Independent states
  - Large action space

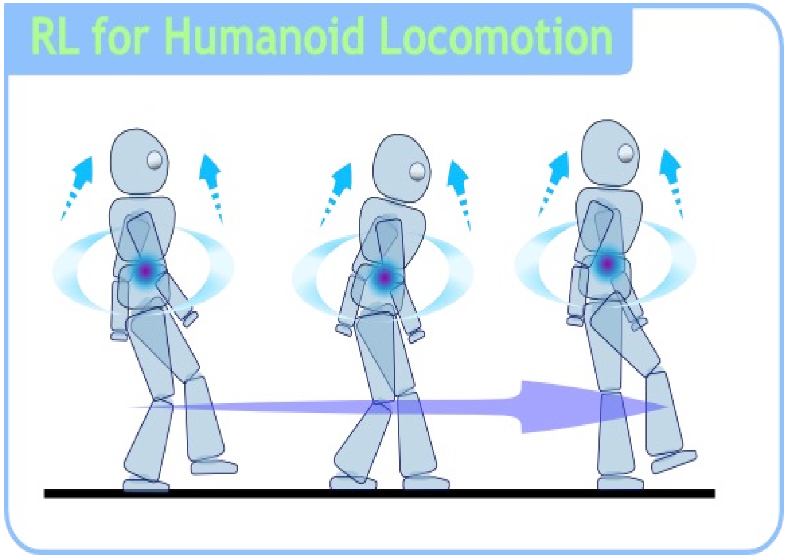






Reality check: dynamic systems



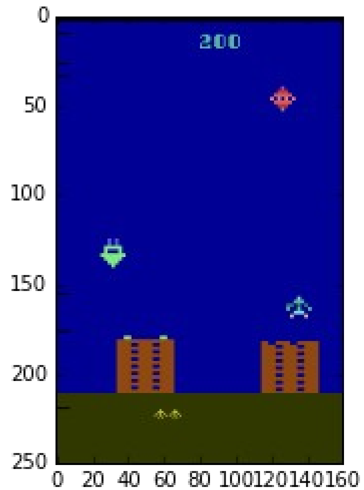


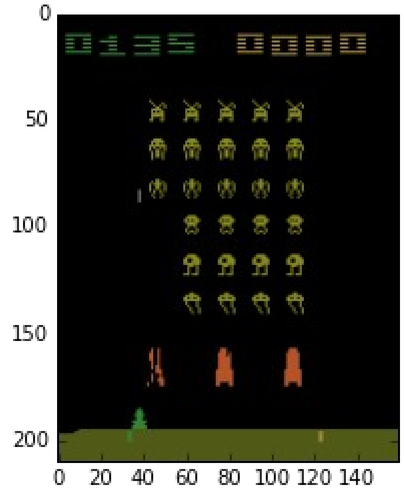
Reality check: MOAR

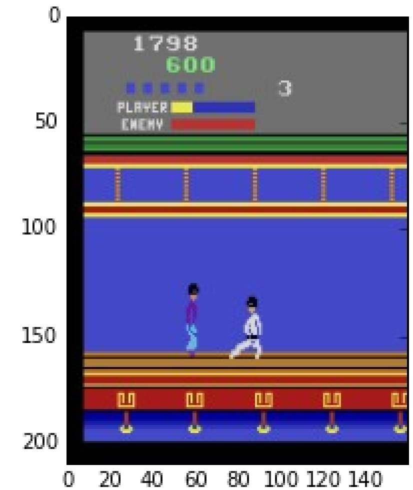
- **Cases:**
  - Robots
  - Self-driving vehicles
  - Pilot assistant
  - More robots!

- **Common traits:**
  - Continuous state space
  - Continuous action space
  - Partially-observable environment
  - LONG sessions

~~Reality~~ check: videogames

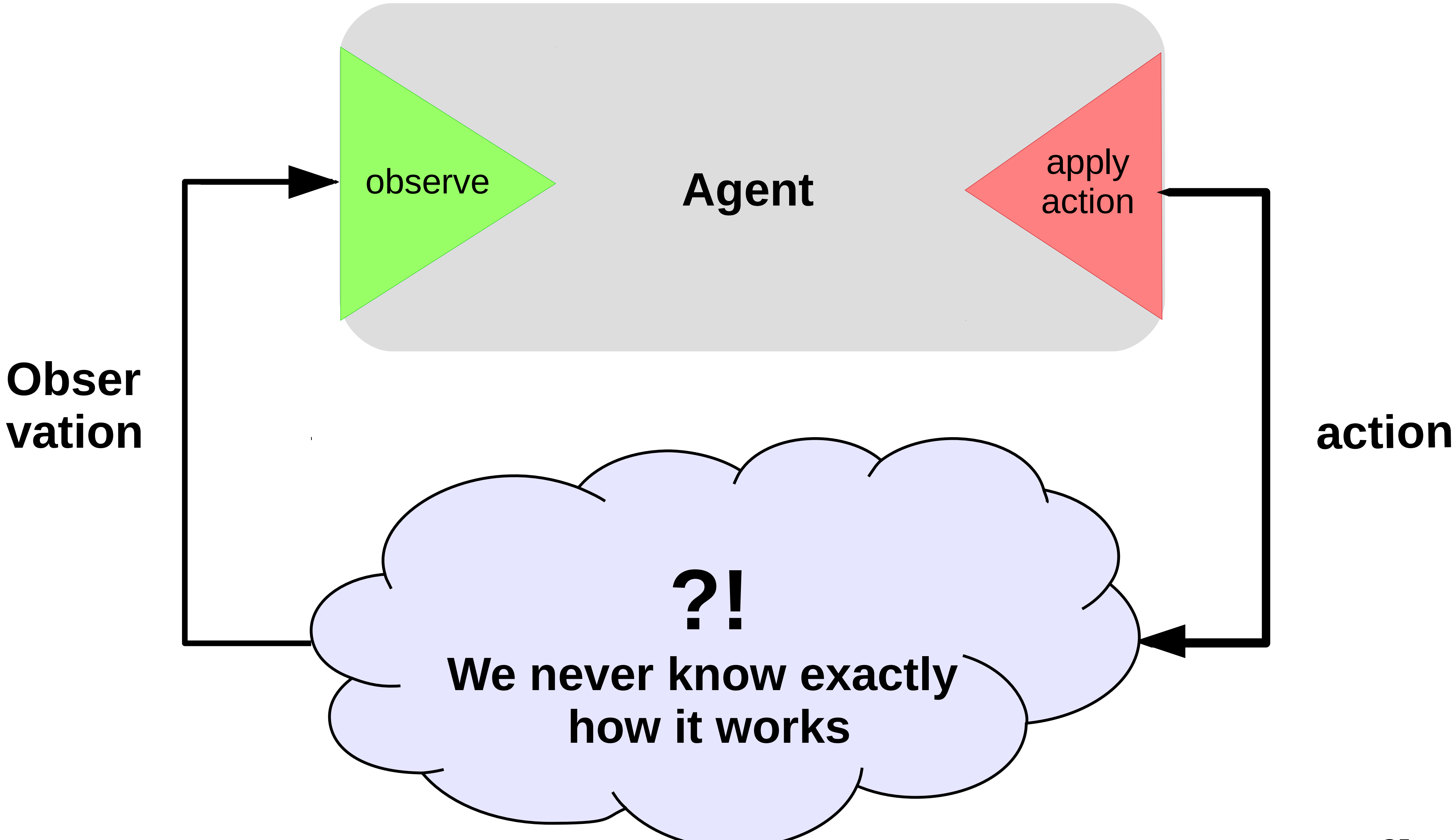






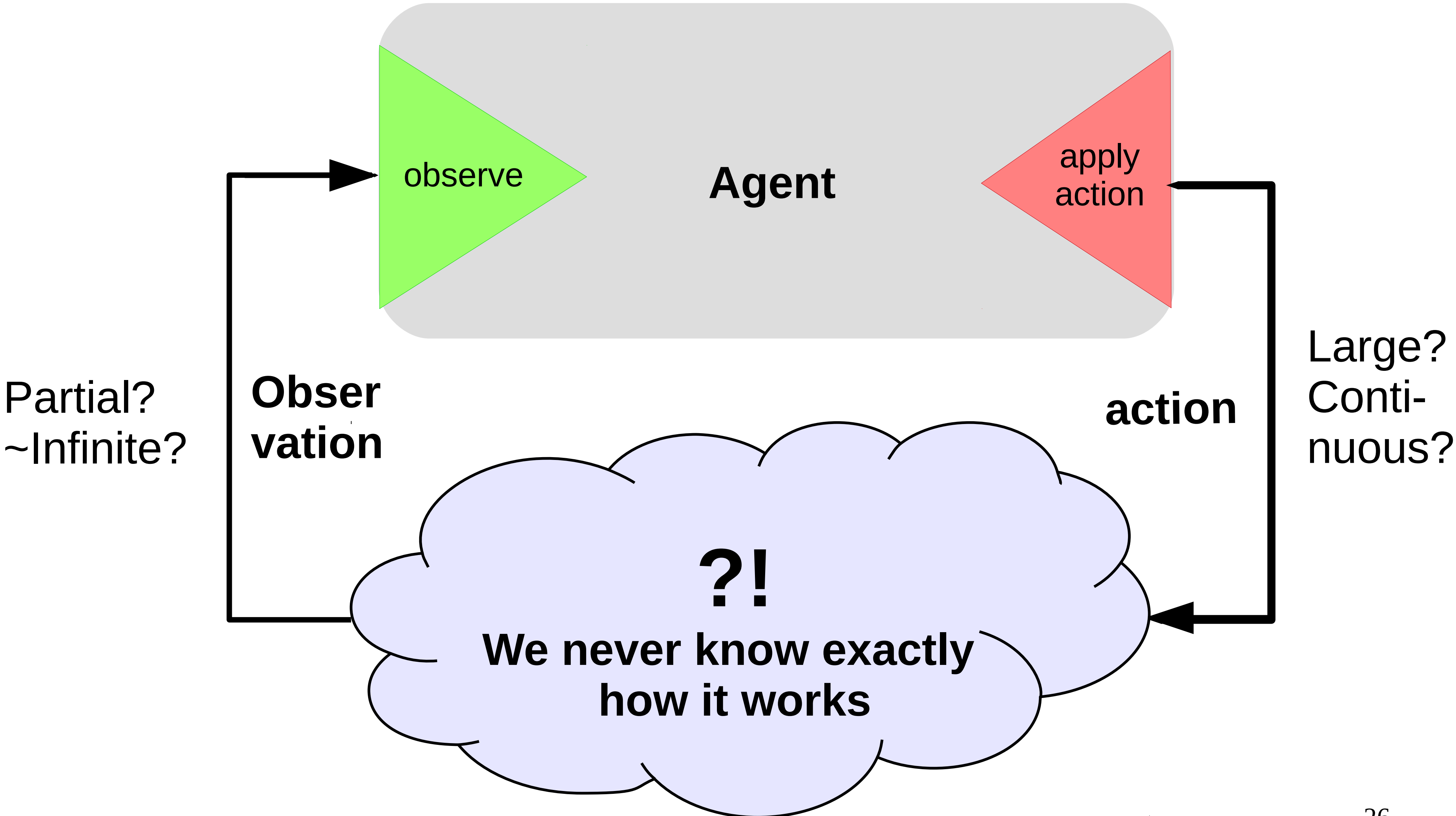
- **Trivia:** What are the states and actions?
  What are the problems?

Real world



observe
Agent
apply
action
Obser
vation
action
?!
We never know exactly
how it works

Real world



observe
Agent
apply action
Obser vation
Partial?
~Infinite?
action
Large?
Conti-
nuous?
?!
We never know exactly how it works

Problem:

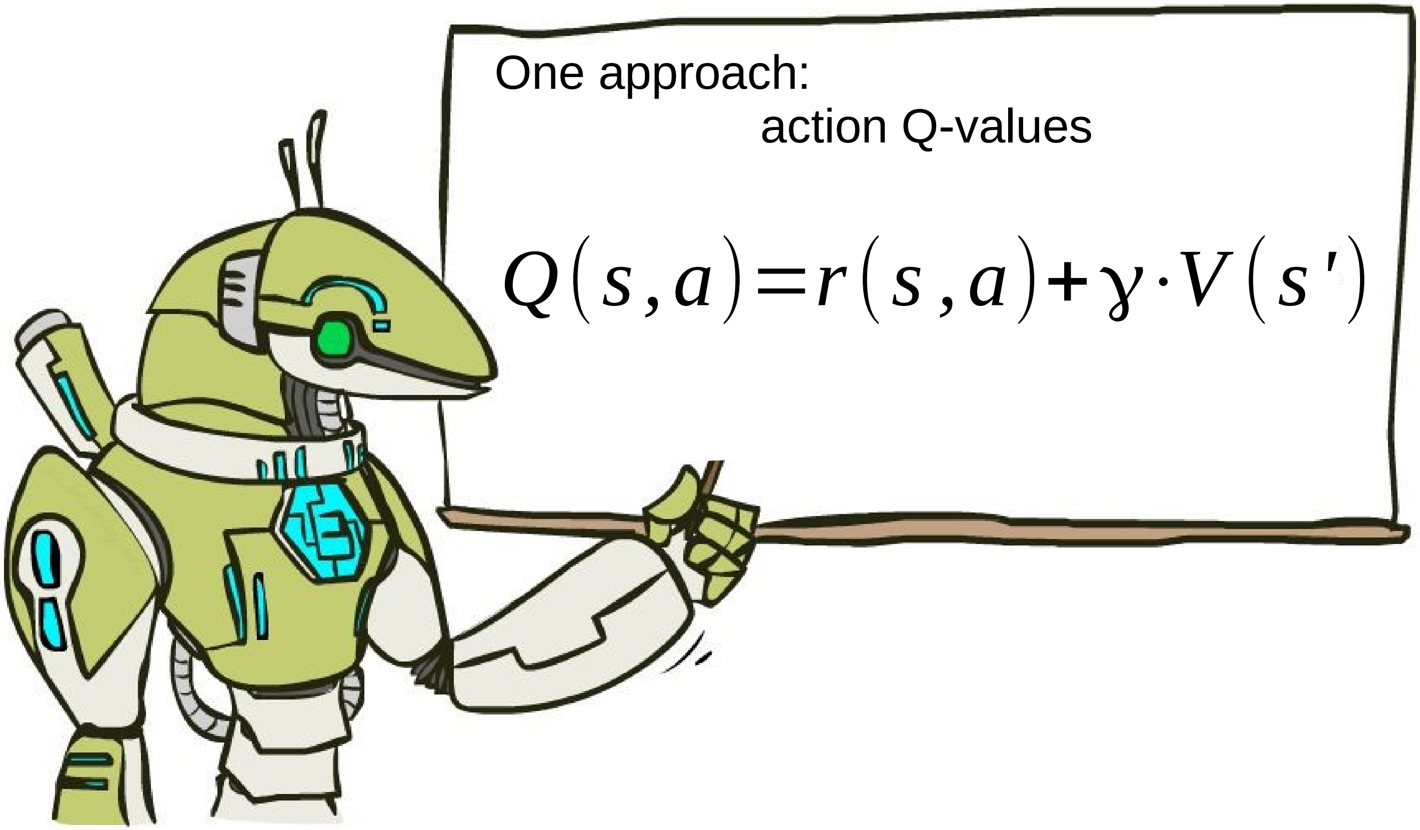
We never know actual

$P(s'|s,a)$

Learn it?

Get rid of it?

From V to Q

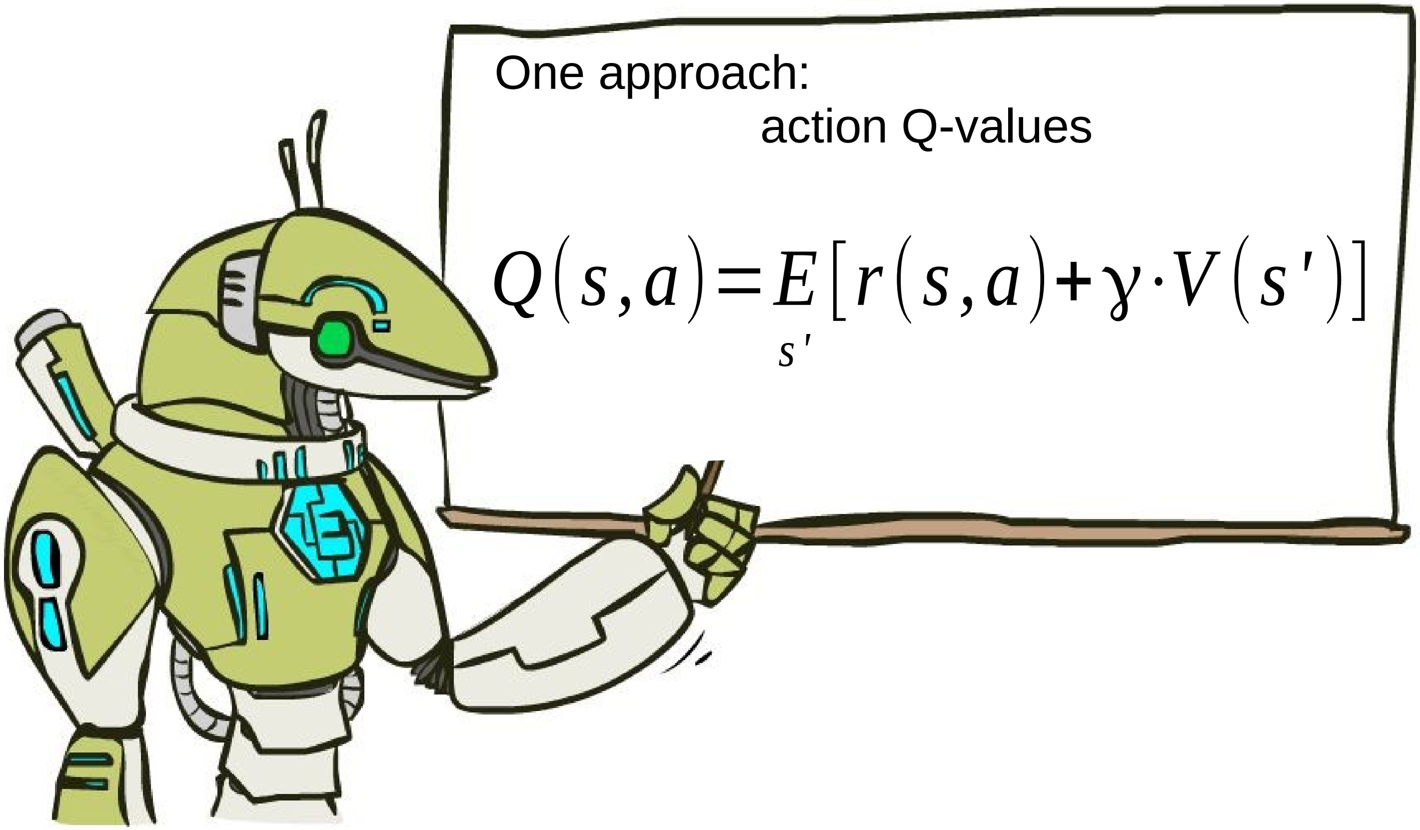


Action value Q(s,a) is the expected total reward **R** agent gets from state **s** by taking action **a** and following policy **π** from next state.

Q(s, a) : state s에서 action a를 취하고, 그 다음부터는 pi 정책을 따라갈 때 R의 기대값

$$\pi(s) : argmax_a Q(s,a)$$

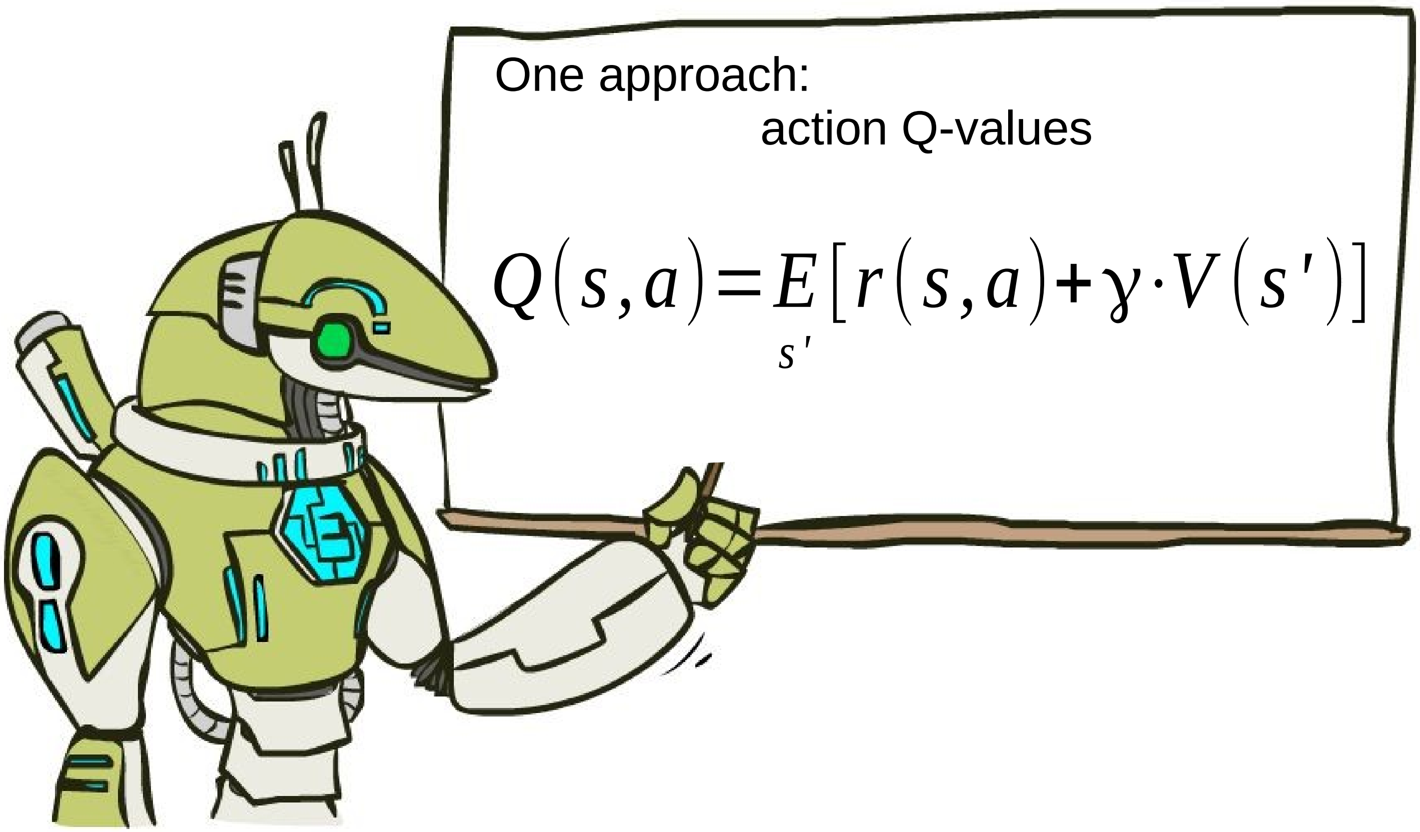
From V to Q



Action value Q(s,a) is the expected total reward **R** agent gets from state **s** by taking action **a** and following policy **π** from next state.

$$\pi(s) : argmax_a Q(s,a)$$

From V to Q

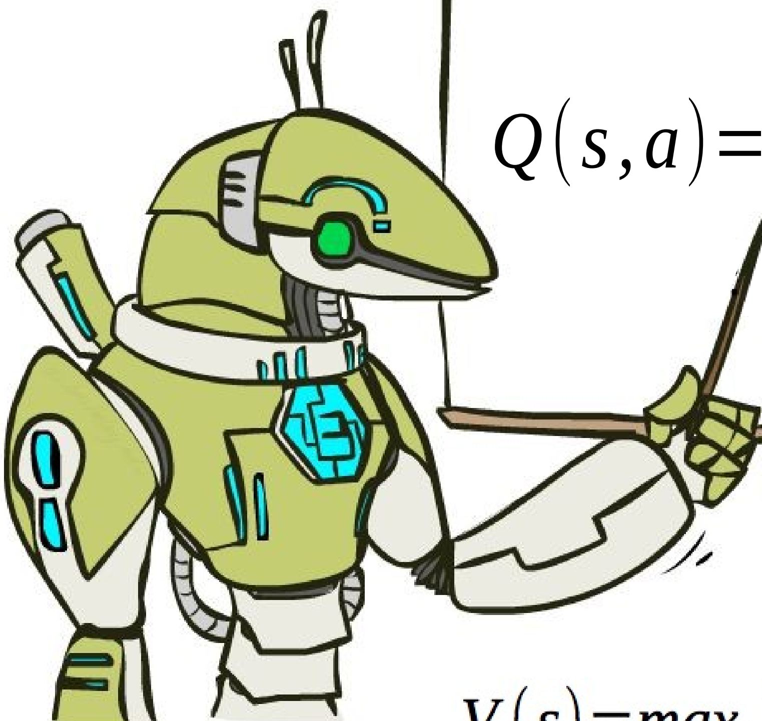


$$Q(s,a) = \underset{s'}{E}\left[r(s,a) + \gamma \cdot V(s')\right]$$

Action value Q(s,a) is the expected total reward **R** agent gets from state **s** by taking action **a** and following policy **π** from next state.

$$\pi(s) : argmax_a Q(s,a)$$

From V to Q



One approach:

action Q-values

$$Q(s,a)=E_{s'}[r(s,a)+\gamma\cdot V(s')]$$

하나의 action a에 대해 sampling

We can replace
P(s' | s,a)
with sampling

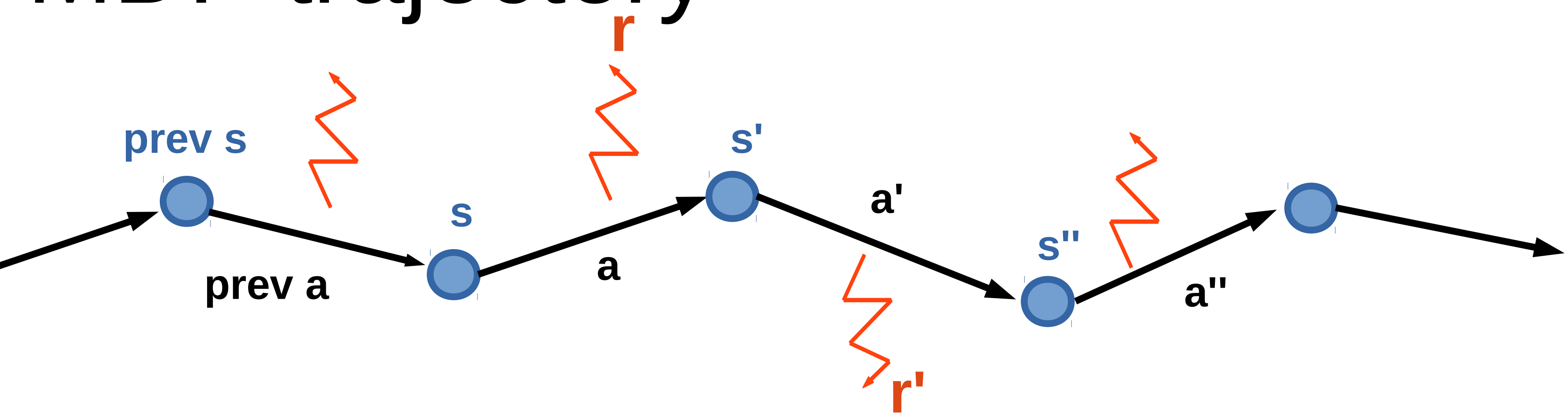
$$V(s)=max_a[r(s,a)+\gamma\cdot E_{s'\sim P(s'|s,a)}V(s')]$$

모든 action a에 대해 sampling해야함..

$$Q(s_t,a_t)\leftarrow\alpha\cdot(r_t+\gamma\cdot max_{a'}Q(s_{t+1},a'))+(1-\alpha)Q(s_t,a_t)$$

$$\pi(s):argmax_a Q(s,a)$$

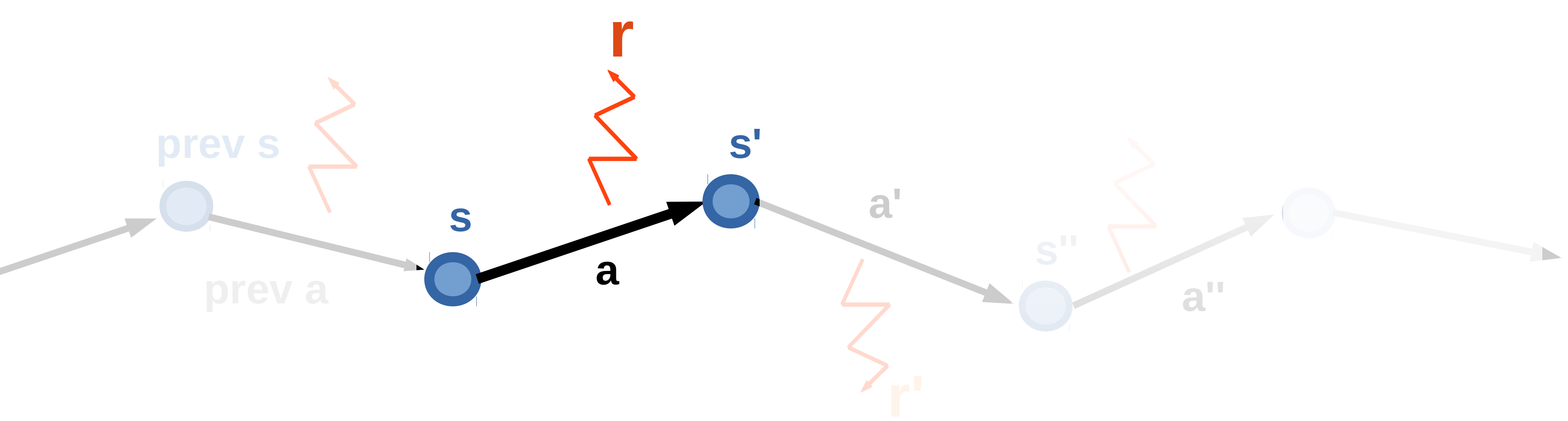
MDP trajectory



- sample sequence of
  - states (s)
  - actions (a)
  - rewards (r)

- Can be infinite, we can't wait that long

Q-learning

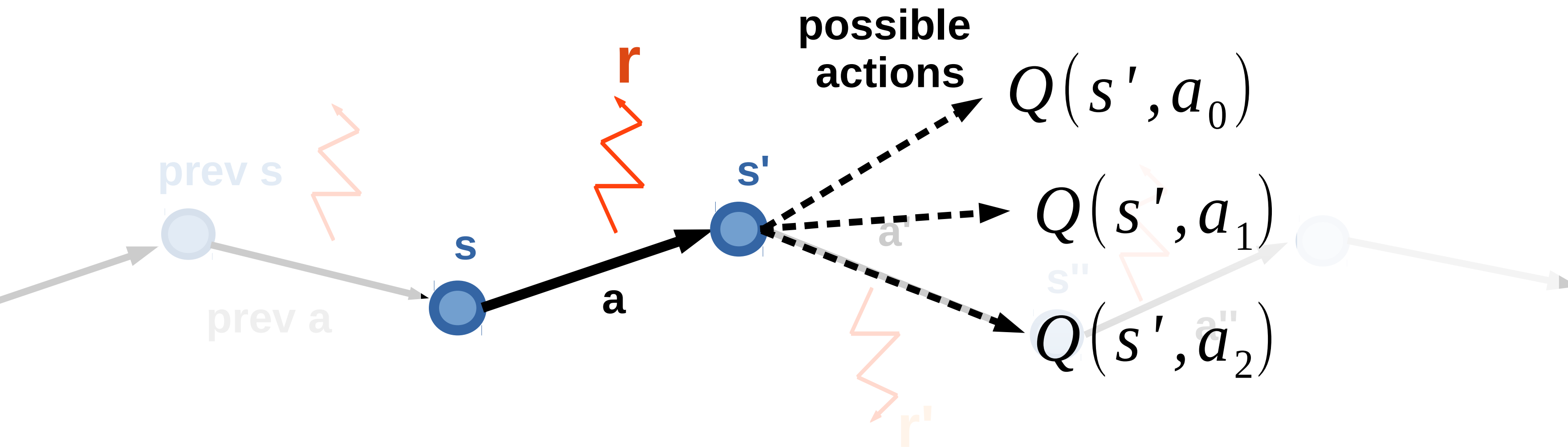


$$\forall s \in S, \forall a \in A, Q(s,a) \leftarrow 0$$

Loop:

- Sample <**s,a,r,s'**> from env

Q-learning



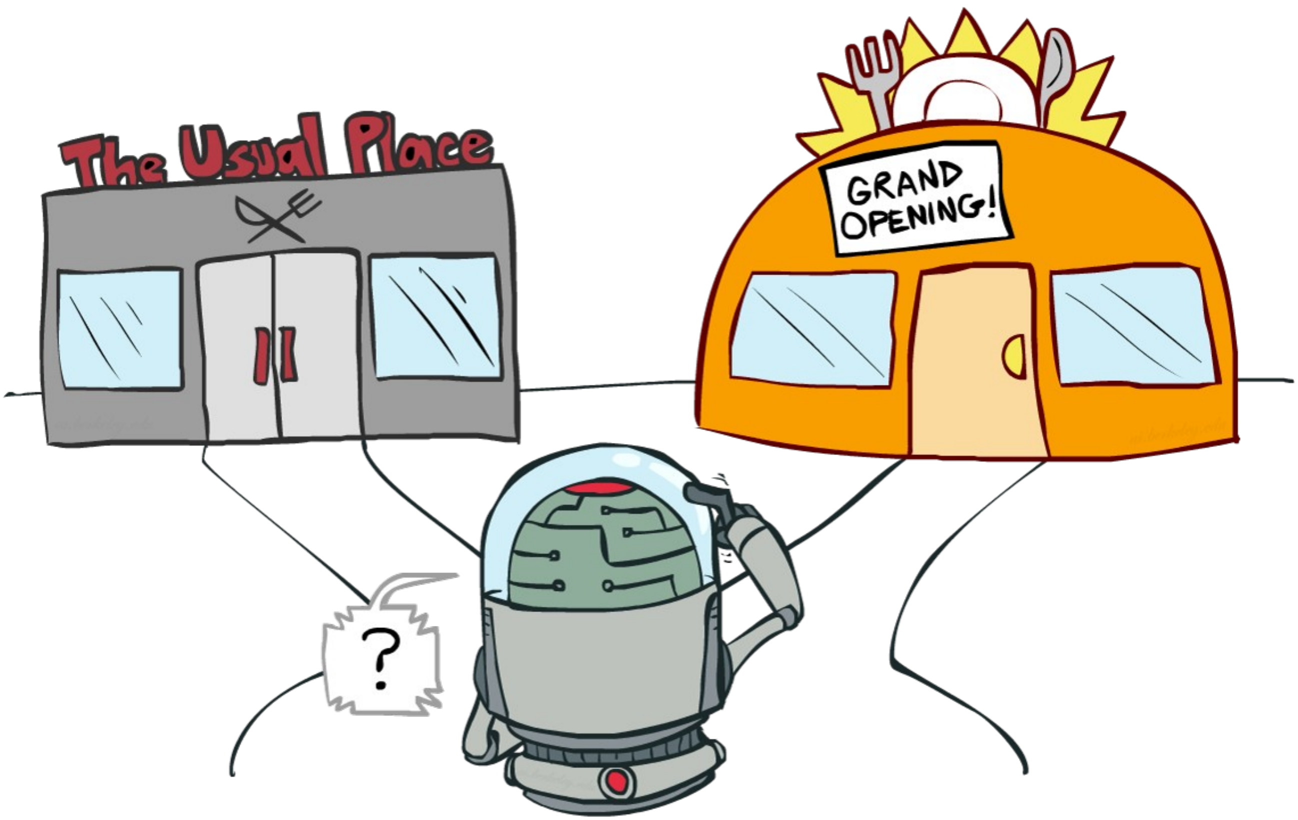
$$\forall s \in S, \forall a \in A, Q(s,a) \leftarrow 0$$

Loop:

- Sample <**s**,**a**,**r**,**s'**> from env
- Compute $\hat{Q}(s,a) = r(s,a) + \gamma \max_{a_i} Q(s',a_i)$
- Update $Q(s,a) \leftarrow \alpha \cdot \hat{Q}(s,a) + (1-\alpha) Q(s,a)$

Exploration Vs Exploitation

Balance between using what you learned and trying to find something even better



Exploration Vs Exploitation

Strategies:

- ε-greedy
  - With probability ε take a uniformly random action; otherwise take optimal action.
- Softmax

  Pick action proportional to softmax of shifted normalized Q-values.

$$P(a) = softmax\left(\frac{Q(a)}{\tau}\right)$$

- Some methods have a built-in exploration strategy (e.g. crossentropy, A2c)

Problem:

State space is usually large,

sometimes continuous.

And so is action space;

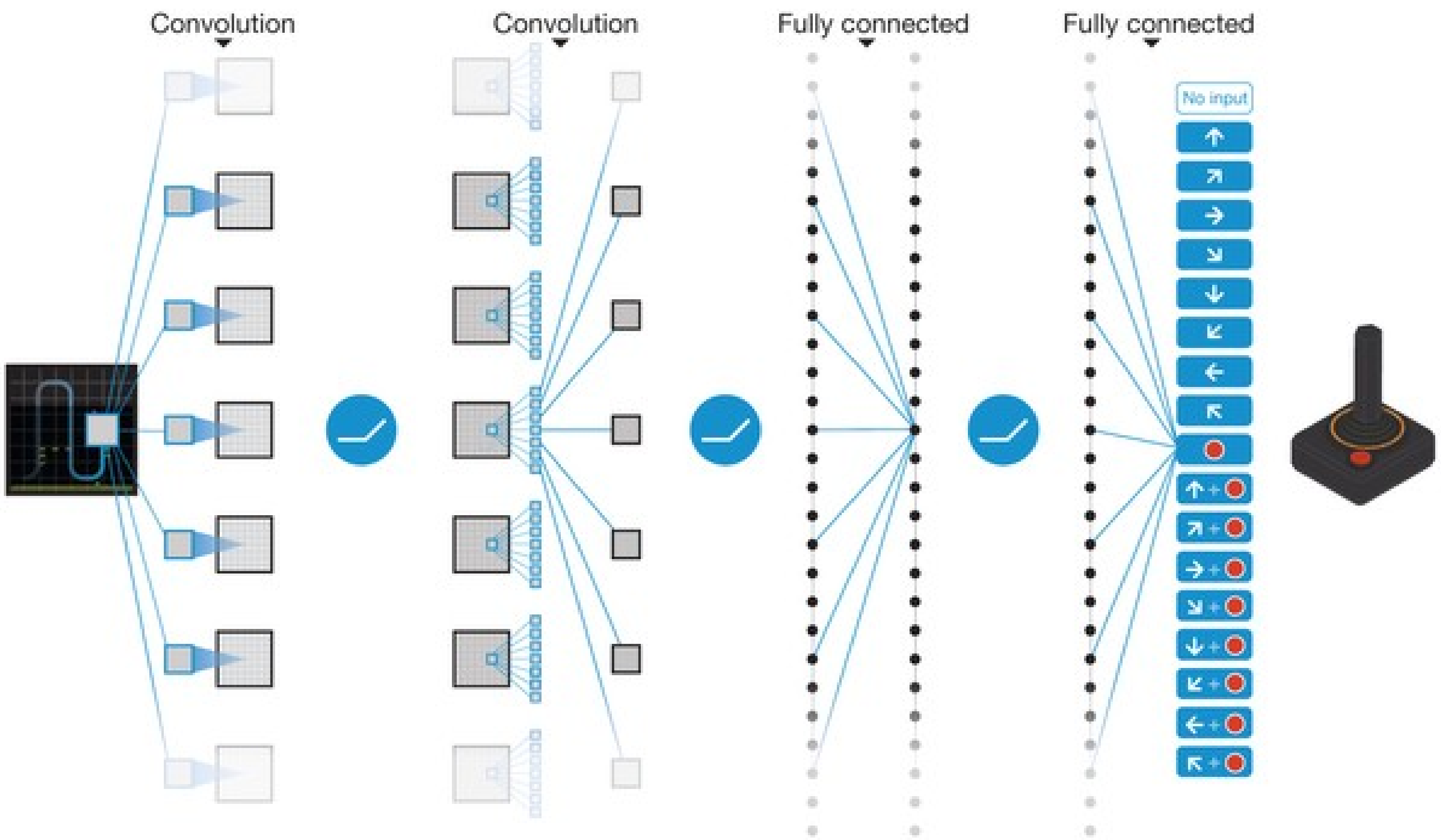
However, states do have a structure, similar states have similar action outcomes.

From tables to approximations

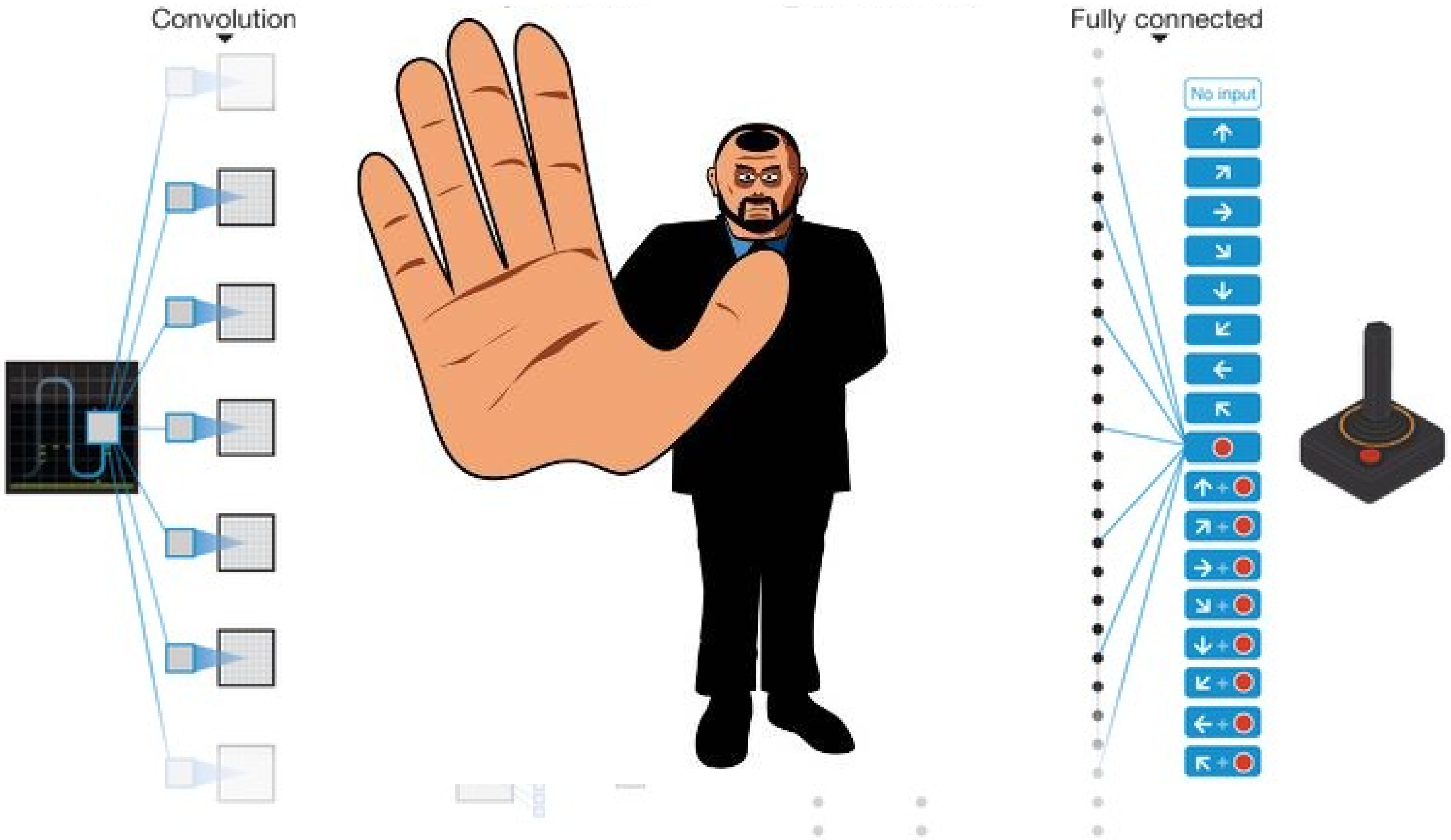
- Before:
  - For all states, for all actions, remember Q(s,a)
- Now:
  - Approximate Q(s,a) with some function
  - e.g. linear model over state features

$$argmin_{w,b}\left(Q(s_t,a_t)-[r_t+\gamma\cdot max_{a'}Q(s_{t+1},a')]\right)^2$$

Smells like a neural network



Not so fast...



Discounted reward MDP

Objective:
Total reward

$$R_t = r_t + \gamma \cdot r_{t+1} + \gamma^2 \cdot r_{t+2} + \ldots + \gamma^n \cdot r_{t+n}$$

$$R_t = \sum_i \gamma^i \cdot r_{t+i} \qquad \gamma \in (0,1)\, const$$

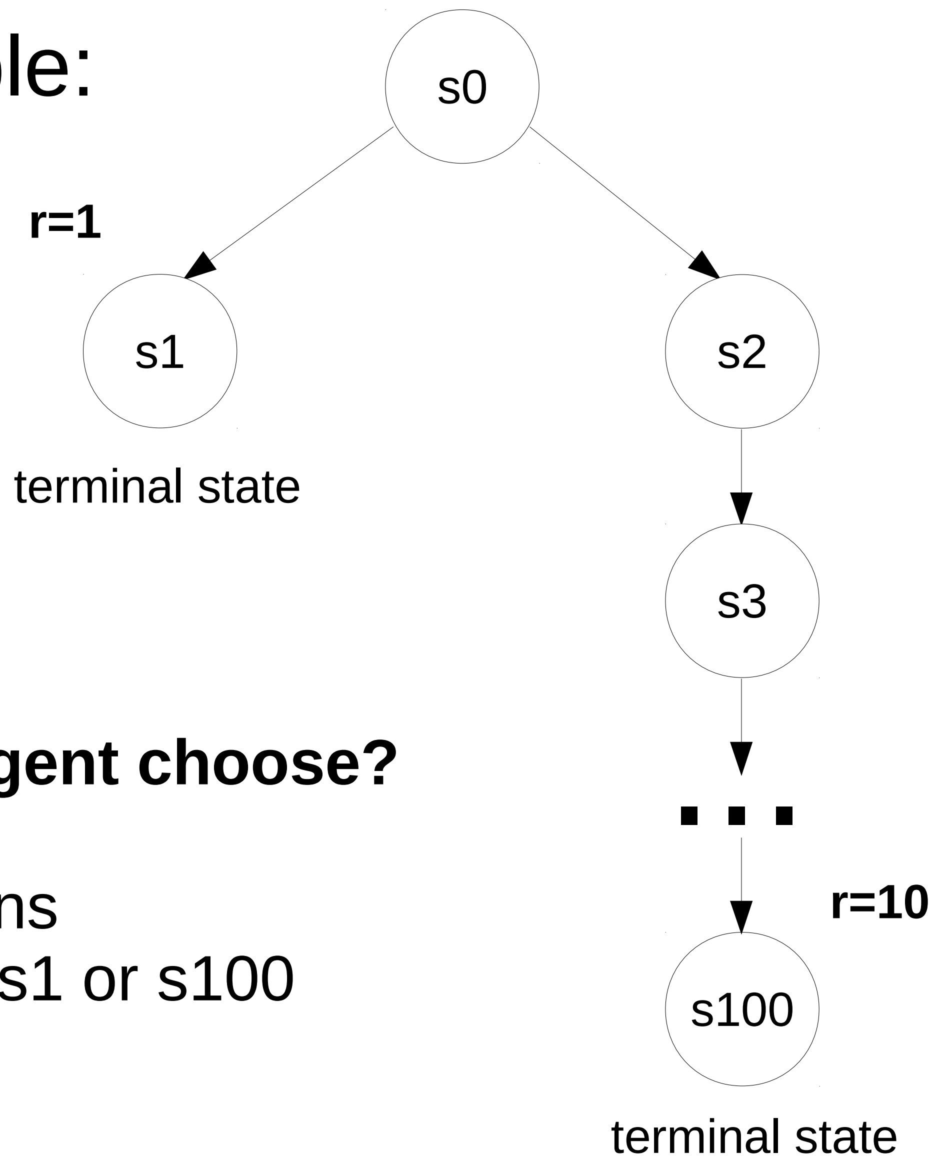
Reinforcement learning:
- Find policy that maximizes the expected reward

$$\pi = P(a|s) : E[R] \rightarrow max$$

Optimal policy isn't always maximizing monte-carlo reward!

Discounted reward fails #1

Trivial example:



What path will agent choose?

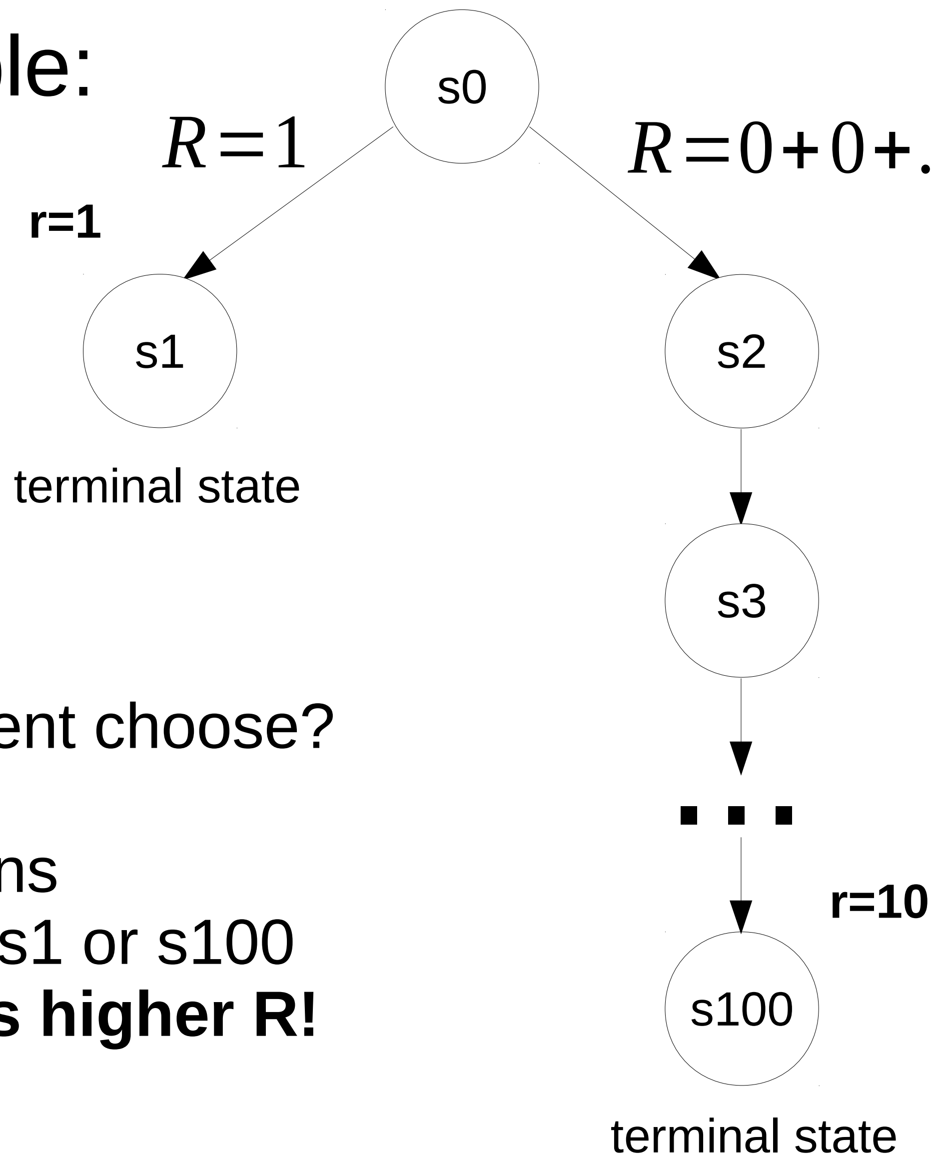
- γ=0.9
- arrows = actions
- game ends at s1 or s100

Discounted reward fails #1

Trivial example:

$$R=1$$

$$R=0+0+\ldots+\gamma^{99}\cdot 10\approx 10^{-4}$$



What path will agent choose?

- γ=0.9
- arrows = actions
- game ends at s1 or s100
- **left action has higher R!**

Discounted reward fails #2

Deephack'17 qualification round, Atari Skiing

- You steer the red guy
- Session lasts ~5k steps

- You get -3~-7 reward each tick
  (faster game = better score)

- At the end of session, you get up to r=-30k
  (based on passing gates, etc.)

- Q-learning with gamma=0.99 fails
  it doesn't learn to pass gates

What's the problem?

http://rl.deephack.me/

Discounted reward fails #2

Deephack'17 qualification round, Atari Skiing

- You steer the red guy
- Session lasts ~5k steps

- You get -3~-7 reward each tick
  (faster game = better score)

- At the end of session, you get up to r=-30k
  (based on passing gates, etc.)

  - Q-learning with gamma=0.99 fails

http://rl.deephack.me/

Discounted reward fails #3

CoastRunner7 experiment (openAI)

- You control the boat
- Rewards for getting to checkpoints
- Rewards for collecting bonuses
- What could possibly go wrong?
- “Optimal” policy video:
  https://www.youtube.com/watch?v=tlOIHko8ySg

https://openai.com/blog/faulty-reward-functions/

Nuts and bolts: MC vs TD

Monte-carlo

- Ignores intermediate rewards
  doesn't need **γ** (discount)
- Needs full episode to learn
  Infinite MDP are a problem
- Doesn't use Markov property
  Works with non-markov envs

Temporal Difference

- Uses intermediate rewards
  trains faster under right **γ**
- Learns from incomplete episode
  Works with infinite MDP
- Requires markov property
  Non-markov env is a problem

Nuts and bolts: discount

- Effective horizon $$1+\gamma+\gamma^2+\ldots=\frac{1}{(1-\gamma)}$$

  Heuristic: your agent stops giving a damn in *this many* turns.

  Typical values:
  - γ=0.9, 10 turns
  - γ=0.95, 20 turns
  - γ=0.99, 100 turns
  - γ=1, infinitely long

  Higher γ = less stable algorithm.
  γ=1 only works for episodic MDP (finite amount of turns).

Nuts and bolts: discount

- Effective horizon $$1+\gamma+\gamma^2+\ldots=\frac{1}{(1-\gamma)}$$

  Heuristic: your agent stops giving a damn in *this many* turns.

- Atari Skiing, reward was delayed by in 5k steps
- γ=0.99 is not enough
- γ=1 and a few hacks works better
- Or use a better reward function

Let's write some code!